
título

Procesamiento masivo de datos mediante Cassandra y Spark

Máster en Sistemas Informáticos Avanzados
Septiembre de 2016

Autor:

Xabier Zabala Barandiaran

Supervisores:

German Rigau i Claramunt
UPV/EHU

Iñigo Etxabe y Beñat Aranburu
Datik Información Inteligente S.L.

eman ta zabal zazu



Universidad
del País Vasco

Euskal Herriko
Unibertsitatea

informatika
fakultatea



facultad de
informática

Agradecimientos

En primer lugar, quisiera expresar mi gratitud a las personas que han posibilitado la concepción y el desarrollo de la presente tesina. Agradezco a German Rigau i Claramunt, supervisor del proyecto de la UPV/EHU, la predisposición mostrada y el asesoramiento ofrecido durante el transcurso del mismo. Doy también las gracias a Iñigo Etxabe y Beñat Aranburu, supervisores del proyecto en Datik Información Inteligente, por poner a mi disposición todos los medios tecnológicos necesarios para llevar a cabo el trabajo y por el trato ofrecido desde el primer día.

Agradecer, cómo no, a mis padres José Javier Zabala y María Pilar Barandiaran el esfuerzo desempeñado para facilitarme, en la medida que les ha sido posible, el camino que he recorrido hasta llegar aquí. Me congratula haber sabido responder satisfactoriamente a la confianza que ellos han depositado en mí. Por todo lo que han hecho y por todo lo que suponen para mí, un beso enorme para los dos.

No me quisiera olvidar de aquellas personas que me han acompañado durante este maravilloso periplo. Doy las gracias a la cuadrilla y a la gente que he tenido la fortuna de conocer durante estos últimos años. Quisiera agradecer especialmente a las personas que en este pasado lustro se han ganado a pulso el privilegio a ser parte importante de lo que me resta de existencia, a los cuales me atreveré a mencionar aún con el temor de dejar a alguna en el tintero: Adrián Núñez, Eider Irigoyen, Goiatz Irazabal, Jaime Altuna, Marta García y Mikel Etxeberria. No hallo palabras para expresar todo lo que os debo.

Por último, pero no por ello menos importante, quisiera evocar a todos los docentes que han tomado parte en mi formación desde aquel Septiembre del 2009 y agradecer a todos ellos el conocimiento compartido y el esfuerzo invertido en mí durante estos años.

Gracias de todo corazón a la gente mencionada en este breve capítulo por haber hecho de mí un mejor profesional y sobre todo una mejor persona, además de darme las fuerzas necesarias para seguir mejorando en ambos aspectos de cara al futuro.

Resumen

Proyecto Final del Máster en Sistemas Informáticos Avanzados. Estudio empírico sobre el rendimiento ofrecido por varias tecnologías emergentes en el campo del Big Data en comparación a una base de datos tradicional a la hora de operar en escenarios que requieren un almacenamiento y procesamiento eficaz de volúmenes masivos de datos.

Para llevar a cabo el experimento, dos entornos de prueba totalmente aislados han sido erigidos sobre la misma máquina física. En el primero, se ha construido un clúster compuesto por tres nodos virtuales que operan en una misma red privada. Dichos nodos han sido dotados de tecnología necesaria para el correcto funcionamiento de Apache Cassandra [1] y Apache Spark [6]. En el segundo entorno, se ha instalado MySQL Server sobre un nodo virtual de potencia equivalente al clúster mencionado con anterioridad. Una vez habiendo poblado las bases de datos mediante un data-set público de aproximadamente 25GB y diseñadas unas consultas acorde a la naturaleza de los datos, se han ejecutado dichas consultas para así cuantificar el tiempo de respuesta en cada escenario.

El estudio evidencia que a la hora de trabajar con volúmenes masivos de datos el binomio entre Apache Cassandra y Apache Spark mejora sustancialmente los tiempos de procesamiento obtenidos con MySQL, además de ofrecer una solución totalmente escalable y tolerante a fallos. No obstante, para gozar de dichas ventajas se antoja necesario un análisis previo de los datos que se desean tratar, aspecto en el que MySQL demuestra ofrecer una mayor libertad.

Palabras Clave: Apache Cassandra, Apache Spark, Big Data, Clustering, Comparativa, NoSQL.

Índice general

1	Introducción	1
1.1	Contexto	2
1.2	Propuesta	3
1.3	Organización del documento	4
2	Análisis de las tecnologías propuestas	7
2.1	Apache Cassandra	7
2.1.1	Funcionamiento	8
2.1.2	Cassandra Query Language (CQL)	11
2.2	Apache Spark	11
2.2.1	Funcionamiento	12
	Bibliografía y Referencias	17

Índice de figuras

1.	Funcionamiento resumido de iPanel	2
2.	Estructura de Cassandra	9
3.	Particionamiento en Cassandra	10
4.	Ecosistema Spark	12
5.	Arquitectura Spark	13
6.	Arquitectura Spark	14

Índice de cuadros

Capítulo 1

Introducción

Desde Aristóteles y su libro Segundos Analíticos ¹ hasta Galileo, padre de la ciencia moderna, adalides del conocimiento han proclamado que un método de investigación basado en lo empírico y en la medición, sujeto a los principios específicos de las pruebas de razonamiento es el camino para conocer la verdad.

Hoy en día, época en la que los avances tecnológico han posibilitado observar y medir de forma exhaustiva un gran abanico de fenómenos, la ingente cantidad de datos que se genera en el proceso es, a veces, intratable usando las tecnologías convencionales, y por ende, es imposible extraer todo el conocimiento que atesoran. El problema, lejos de atenuarse, se acrecienta con el paso del tiempo, ya que, estudios como el realizado por McKinsey Global Institute estiman que el volumen de datos que se genera crece un 40 % cada año y auguran que entre 2009 y 2020 se verá multiplicado por 44 [2].

Por ello, en los últimos años ha irrumpido la necesidad de hallar nuevas metodologías y tecnologías que permitan procesar y extraer el conocimiento que atesora el torrente de información en la cual se encuentra envuelta la sociedad, dando como resultado el nacimiento del Big Data.

El mundo empresarial, por su parte, no se ha mantenido al margen de esta gran revolución. Conscientes de los beneficios que les puede reportar en diferentes aspectos de su negocio, la gran mayoría de las empresas se han interesado en este fenómeno. De un estudio realizado entre los altos ejecutivos de las firmas que lideran el Wall Street se desprende que el 96 % tiene planeadas ciertas iniciativas relacionadas con el Big Data, y el 80 % ya tiene finalizada alguna [5].

¹Órganon II de Aristóteles: Segundo Analíticos se encuentra recopilado en él

1.1. Contexto

Datik Información Inteligente² es una empresa tecnológica perteneciente al Grupo Irizar³ que desarrolla soluciones ITS destinadas a la gestión del transporte, tanto ferroviario como por carretera y movilidad ciudadana.

Uno de los productos estrella de la entidad es el denominado iPanel, concentrador de información que ofrece al operador de transporte servicios de valor añadido en la gestión de la información generada por su flota. El funcionamiento del servicio se puede resumir mediante la Figura 4:

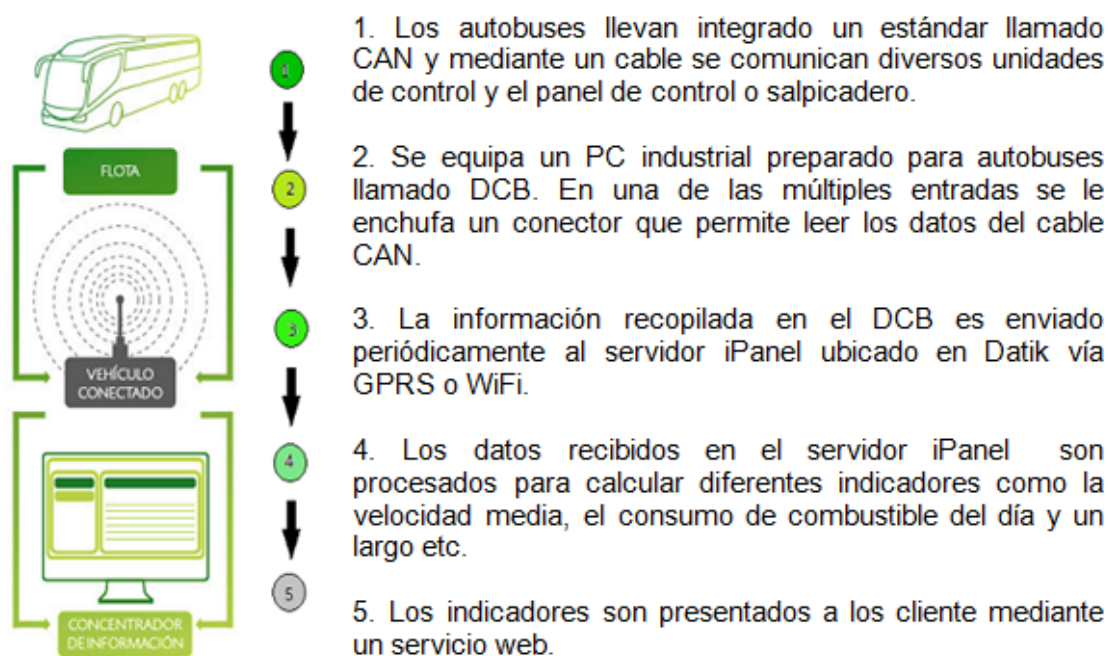


Figura 1: Funcionamiento resumido de iPanel

La incesante integración de nuevos vehículos a iPanel ha generado un crecimiento exponencial en el número de registros almacenados en ciertas tablas de MySQL. Aunque el volumen actual no suponga riesgo alguno para el funcionamiento del servicio, Datik tiene identificados varios escenarios en los que la situación se podría revertir, causando graves problemas en el sistema.

²<http://www.datik.es/>

³<http://www.irizar.com/irizar/>

El primero de todos, es la corrupción de datos. Este fenómeno sucede debido a un bug, fallo de almacenamiento inesperado, o una caída de MySQL cuando el resultado del checksum de una página es diferente al esperado. Aunque ocurre de forma esporádica, compromete seriamente la información que Datik ofrece a sus clientes mediante la aplicación web.

Otro de los problemas, intrínseco a depender de una base de datos centralizada, es el operar sobre un único punto de fallo. Debido a que la mayoría de procesos confluyen en ella, el bloqueo o la caída causada por un servicio puede acarrear la de otros, a priori, totalmente independientes. Para solventar el problema Datik ha optado por migrar su infraestructura a una basada en Microservicios [3], logrando de esa manera, el aislamiento total de los componentes que conforman su ecosistema.

El último problema, es el referente al proceso denominado Cálculo de Indicadores, el cual se ejecuta una vez al día para realizar operaciones aritméticas sobre diversas tablas y después, agrupar los resultados en base a diferentes criterios. Siendo dichas tablas las que mayor crecimiento experimentan, el aumento del volumen de las mismas incrementa de forma desorbitada el tiempo necesario para finalizar el cálculo, pudiendo, en un futuro, llegar a tardar mas de 24 horas y cancelar los indicadores que ofrecen información del último día.

El objetivo del presente proyecto es proponer soluciones a los problemas planteados en este apartado, haciendo especial hincapié en el proceso Cálculo de Indicadores, ya que afecta de forma directa a la información que consumen los clientes de Datik vía iPanel.

1.2. Propuesta

La problemática que envuelve al Cálculo de Indicadores es originado por el incremento exponencial de datos que dicho proceso ha de tratar. No sería descabellado pensar que la solución pueda pasar por escalar la máquina verticalmente y afinar la configuración de MySQL. No obstante, ambas mejoras tienen un límite, mientras que el volumen de los datos seguirá aumentando de forma inexorable, volviendo, tarde o temprano, a tener que lidiar con los problemas del principio.

Siendo imposible reconducir la situación usando las tecnologías tradicionales, en el presente proyecto se propone realizar un cambio de paradigma que implica migrar las tablas relacionadas con los indicadores a un modelo distribuido que posibilite escalar la infraestructura horizontalmente. A su vez, se sugiere dividir el problema en dos apartados, almacenamiento y procesado, dotando la infraestructura de tecnología adecuada

para ofrecer una respuesta eficaz a cada una de las partes.

Debido a que los datos relacionados con el calculo de indicadores no presentan relacion alguna con otro tipo de entidad, se ha

En cuanto al almacenamiento se refiere

Debido a que los datos relacionados con los indicadores datos no relacionales, nosql parece buena alternativa

Analizando el escenario presentado,

** Los datos con los cuales se hara a prueba no son de datik, por motivos de confidencialidad de cantidad

Apache Cassandra es una base de datos distribuida no-sql. Gracias a naturaleza distribuida ayuda a resolver,

Funcionalidades nuevas que trabajan con videos etc

(testear las tecnologias con dataset publico, los datos de la empresa son confidenciales)

Debido a la falta de datos se ha utilizado un dataset publico para emular las condiciones de futuro con las que se va a encontrar datik

1.3. Organización del documento

En esta memoria se ha documentado el desarrollo de la herramienta *exerClick*, dentro del Trabajo de Fin de Grado (TFG) del autor. En el documento se describe la propuesta, la planificación y gestión que esta lleva consigo, la implementación llevada a cabo y las conclusiones finales.

En este primer capítulo se ha introducido el problema a resolver y se ha explicado la propuesta presentada en este proyecto.

En el capítulo 2 se presenta el Documento de Objetivos de Proyecto (DOP). Este recoge el alcance y las fases y tareas del proyecto, el análisis de riesgos y el análisis de factibilidad.

Una vez en el capítulo 3 se explica la gestión llevada a cabo durante el proyecto. Se presentan las metodologías utilizadas: Metodologías Ágiles e InterMod (adaptada a las necesidades de este proyecto). A continuación se detallan cada una de las iteraciones llevadas a cabo (como parte de la metodología InterMod): duración, objetivos y tareas realizadas. Al final del capítulo se muestra la documentación asociada a las iteraciones y los objetivos, además del seguimiento de tiempo realizado.

A continuación, en el capítulo 4 se detalla el análisis de requisitos. Primero se detallan los requisitos no-funcionales y luego los funcionales (prototipos en papel llevados a cabo

durante las primeras iteraciones que dan una visión global del proyecto).

En el capítulo 5 se explica el diseño e implementación llevados a cabo. Se comienza mostrando la estructura de documentos del proyecto, luego el diseño realizado en base al análisis de requisitos del capítulo 4 y finalmente una visión general de la implementación de la lógica de negocio.

Para finalizar, en el capítulo 6 se presentan las conclusiones, líneas futuras para el proyecto y las lecciones aprendidas.

Fuera de la estructura general de la memoria, tenemos la bibliografía y los apéndices. En estos últimos tenemos las actas de reuniones, las actas de pruebas y la vista de relaciones de la base de datos (de la parte utilizada o creada específicamente para el proyecto).

Capítulo 2

Análisis de las tecnologías propuestas

2.1. Apache Cassandra

Apache Cassandra es una base de datos distribuida desarrollada por DataStax ¹ bajo la licencia de Apache ² que permite operar sobre grandes volúmenes de datos del tipo clave/valor. Se caracteriza por ofrecer una disponibilidad total y mayor escalabilidad lineal en comparación a otras bases de datos NoSQL, utilizando, para ello, una serie de nodos homogéneos que se comunican mediante un protocolo P2P de replicación asincrónica, lo cual permite realizar operaciones de baja latencia para todos los clientes sin necesidad de un servidor maestro.

En 2012, investigadores de la Universidad de Toronto que estudian los sistemas NoSQL concluyeron que "En términos de escalabilidad, hay un claro ganador a través de nuestros experimentos. Cassandra logra el mejor rendimiento para el número máximo de nodos en todos los experimentos"[4].

Actualmente está siendo utilizada por muchas de las aplicaciones en negocios modernos, siendo la base de datos elegida por un cuarto de las empresas de la Fortune 100[.]. Claro ejemplo de ello son empresas mundialmente conocidas como Apple, Facebook o Netflix, los cuales utilizan Cassandra como parte de su entramado tecnológico desde hace ya unos años. Cabe destacar que su uso no se limita al mundo empresarial. Muestra de ello es la acogida que ha tenido en el ámbito de la investigación, formando parte en experimentos punteros a nivel mundial como algunos de los realizados en el CERN[.].

¹<http://www.datastax.com/>

²<http://www.apache.org/>

Es indispensable tener en cuenta que no se trata de una base de datos de propósito general, por lo que, es de vital importancia conocer cuando se va a poder expresar su potencial al máximo y cuando no.

Será **recomendable** utilizar Cassandra si:

- se desea que la configuración, mantenimiento y el código sea sencillo.
- se necesitan velocidades muy altas en lecturas y escrituras aleatorias.
- no se necesitan múltiples índices secundarios
- existe alta flexibilidad en la estructura de los datos
- se busca una escalabilidad masiva
- se busca alta disponibilidad

No será **recomendable** utilizar Cassandra si:

- se manejan datos relacionales
- hay transacciones de por medio
- se requiere autorización para acceder a datos
- se necesita latencia baja

2.1.1. Funcionamiento

Debido al modelo distribuido sobre el cual esta desarrollado, el primer paso para comprender el funcionamiento de Apache Cassandra es familiarizarse con los conceptos básicos de los sistemas distribuidos.

El Teorema de Brewer[], también conocido como Teorema CAP , enuncia que es imposible para un sistema de cómputo distribuido garantizar simultáneamente las tres propiedades que se presentan a continuación, solo pudiendo cumplir dos de ellas al mismo tiempo, y acabar cumpliendo el restante tarde o temprano.

- **Consistencia**(Consistency): Todos los nodos ven la misma información al mismo tiempo.
- **Disponibilidad**(Availability): La garantía de que cada petición a un nodo reciba una confirmación de si ha sido o no resuelta satisfactoriamente.

- **Tolerancia al Particionado**(Partition Tolerance): El sistema sigue funcionando a pesar de que haya sido partido por un fallo de red.

Para una base de datos distribuida que promete la disponibilidad completa, como lo es Cassandra, el Teorema de Brewer implica la imposibilidad de garantizar la consistencia total de los datos y la necesidad de esperar un tiempo indeterminado para que las réplicas de un registro modificado se actualicen correctamente.

Dicha actualización corre a cargo de un protocolo Gossip(), encargado del funcionamiento coordinado de la infraestructura. Mediante paso de mensajes periódicos da a conocer el estado de un nodo al resto de sus vecinos, posibilitando de esa manera, que cada nodo pueda mantener actualizadas sus réplicas, además de conocer en cada instante qué nodo está caído y cual no.

No obstante, existen formas de minimizar el tiempo que hace falta para alcanzar un estado consistente. La primera, es proveer la infraestructura de medios físicos necesarios para acelerar el intercambio de datos entre los nodos y evitar que la red se congestione en el proceso. La segunda, trata sobre la posibilidad que Cassandra ofrece de elegir el nivel de consistencia() con el que se ejecuta cada consulta.

Analizando las características propias de Cassandra y en concreto su estructura, la Keyspace es un espacio de nombres para un conjunto de ColumnFamily. Por lo general, se utiliza uno por aplicación y es considerado como el equivalente a una base de datos del modelo relacional. El ColumnFamily, a su vez, es capaz de almacenar diferentes columnas, siendo el homólogo de una tabla del modelo relacional. Para finalizar, una columna está compuesta por clave y valor, además de un campo del tipo timestamp gracias al cual se actualizan las réplicas obsoletas.

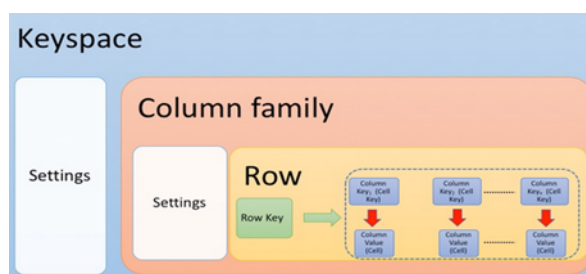


Figura 2: Estructura de Cassandra

A la hora de definir un keyspace dos atributos estrechamente relacionados han de ser especificados: Replication Factor y Replica Placement Strategy(). El primero, consiste en un número que indica cuantas copias de un mismo registro se han de almacenar en la

infraestructura. Cada nodo posee una réplica para un cierto rango de datos y si uno de ellos falla, otro que posea dicha réplica puede responder a la petición sin tener que interrumpir el servicio. El segundo, define cómo se han de repartir los registros replicados por el anillo, ofreciendo distintas opciones dependiendo si los nodos del clúster se encuentran alojados en un único datacenter o repartidos entre varios.

El atributo Replica Placement Strategy cobra especial interés al operar sobre un clúster cuyos nodos están distribuidos en punto geográficos lejanos, ya que permite almacenar cada réplica en diferentes puntos del globo. De esa manera, se consigue minimizar la latencia de las peticiones recibidas desde diferentes puntos del planeta y además, proteger la base de datos de una posible caída a nivel de datacenter debido a diversos catástrofes.

Si un componente de Cassandra tiene especial importancia en su funcionamiento es el ColumnFamily. En una base de datos relacional como MySQL, las tablas son diseñadas con el objetivo de minimizar la redundancia de los datos almacenados en ella, siguiendo para lograr dicho objetivo un proceso de normalización[1]. En cambio, al operar con Cassandra, ocurre exactamente lo contrario. Las tablas se construyen buscando una respuesta rápida a las consultas sin importar que ello implique tener que almacenar datos redundante en la base de datos.

partition key etc

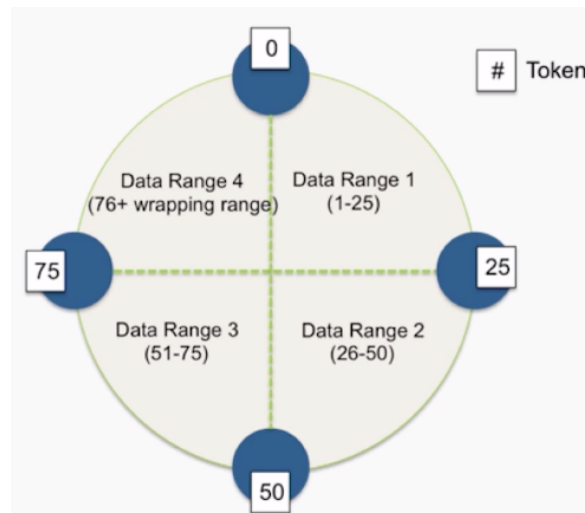


Figura 3: Particionamiento en Cassandra

consecuencia: restriccion en las consultas que se pueden realizar

2.1.2. Cassandra Query Language (CQL)

permite conectarse al cualquier nodo del cluter (homogeneidad)

Apache Cassandra posee su propio lenguaje de consultas, el denominado Cassandra Query Language (CQL). Su sintaxis guarda una gran similitud con la de SQL, lo cual facilita, de forma notoria, el salto que supone pasar de trabajar con bases de datos relacionales a distribuidos.

Aún siendo sintácticamente tan parecido a SQL, presenta ciertas restricciones debido a que es un lenguaje de consultas de una base de datos distribuida. Por ejemplo, no ofrece la posibilidad de realizar operaciones como JOIN y es totalmente necesario especificar todas los atributos que componen la clave primaria a la hora de realizar cualquier consulta de filtrado o de actualización en la tabla. La única operación que no cumple esta restricción es un select que contenga la clausula where, ya que, al definir la clave primaria Cassandra indexa de forma automática todos sus componente, posibilitando más tarde hacer uso de ellos en este caso concreto.

Otra de las peculiaridades que presenta CQL es el hecho de ofrecer dos modos distintos de realizar un update. El primero de todos es el mencionado en el párrafo anterior. El segundo posibilita actualizar una columna realizando un insert repitiendo el valor de las claves primarias de una columna ya existente en la base de datos. Esta segunda forma es cómoda a la par de peligrosa porque Cassandra no notifica si una clave primaria ya existe en la base de datos o no, pudiendo un insert desencadenar en un update no deseado.

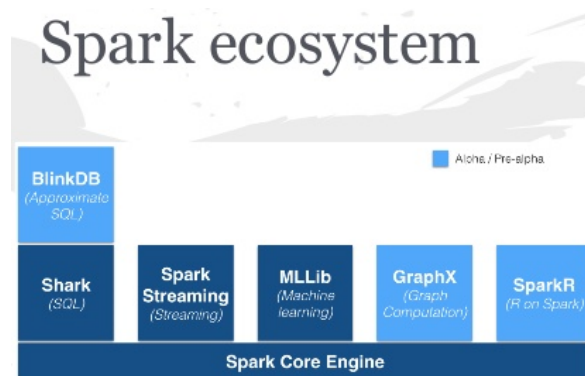
2.2. Apache Spark

Apache Spark[9] es un proyecto open source de computación en clúster. Desde el principio fue diseñado para poder ejecutar algoritmos iterativos en memoria sin la necesidad de almacenar en disco los resultados intermedios generados durante el proceso. Esta peculiaridad permite que los procesamientos llevados a cabo con Spark puedan llegar a ser, en algunos casos concretos, 100 veces más rápidos que los de MapReduce[10].

A mediados de 2014, coincidiendo con el lanzamiento de la primera versión, alcanzó la cifra de 465 colaboradores, convirtiéndolo en el proyecto más activo entre los relacionados con el Big Data dentro de la Apache Software Foundation.

Apache Spark está compuesto por múltiples y variados componentes que pueden ser utilizados de forma conjunta.

La base del proyecto es el denominado Spark Core. Proporciona envío distribuido de tareas, planificación y funciones básicas de entrada salida. La abstracción fundamental

**Figura 4: Ecosistema Spark**

de programación se llama Resilient Distributed Datasets (RDD)[11], una colección lógica de datos particionados a través de las máquinas que se expone mediante una API integrada en lenguajes como Java, Python y Scala.

2.2.1. Funcionamiento

Para el funcionamiento de Spark, es condición sine qua non que los nodos de la infraestructura tengan acceso a la totalidad de los datos que se desea tratar. Ello implica que para procesar un fichero de 50GB, cada nodo tendría que poseer una copia del mismo almacenado en su disco. Esta praxis es inviable, ya que más allá de los problemas de consistencia que generaría, para nada es eficiente ocupar la memoria de todos los nodos con información redundante y procesar el fichero entero cuando en realidad se va a hacer uso de una pequeña porción de dichos datos en cada ejecución.

Las bases de datos distribuidas como Cassandra solventan los problemas anteriormente mencionados. Se encargan de distribuir los datos entre diferentes nodos del clúster, ofrecen la posibilidad de acceder a ellos desde cualquier punto y mantienen la consistencia de los mismos a cambio de sufrir una pequeña latencia en el caso de requerir información almacenada en otro nodo de la infraestructura.

Al ejecutar una aplicación que opera con Spark, un componente denominado driver es lanzado. Debido a la necesidad de obtener recursos (CPU y memoria) para llevar a cabo la computación que se le ha encomendado, se comunica con un nodo del clúster que, mediante especificación previa, adopta el rol de maestro. Éste pregunta a todos los nodos que conforman la infraestructura sobre la cantidad de recursos disponibles que poseen y así asignarles los executors correspondiente. Las máquinas que alojen al menos

un ejecutor pasan a denominarse worker y a partir de este momento, cada ejecutor podrá comunicarse directamente con el Driver para poder recibir las tareas que éste le envíe.

Un ejecutor es una unidad de trabajo que se encarga de computar las tareas que le encomienda el driver. El número de executors que puede albergar cada worker está directamente relacionado con el número de procesadores que este posee. De la misma forma, es posible repartir la memoria RAM que dispone el nodo worker entre varios executors. Spark permite modificar ambos parámetros programáticamente permitiendo así poder amoldarse a las particularidades de cada ejecución.

Para transferir el código del programa, residente en la máquina del driver, éste adopta el rol de servidor e intenta enviar dicho código a los workers. Si el fichero JAR que contiene el código ha sido recibido correctamente por sus destinatarios, estos responden mediante un ACK y en caso contrario, se vuelve a intentar el envío un número determinado de veces. Una vez llegado al máximo de reintentos, el worker que no haya enviado el ACK es considerado como caído, quedando los executors que albergaba fuera del posterior reparto de tareas.

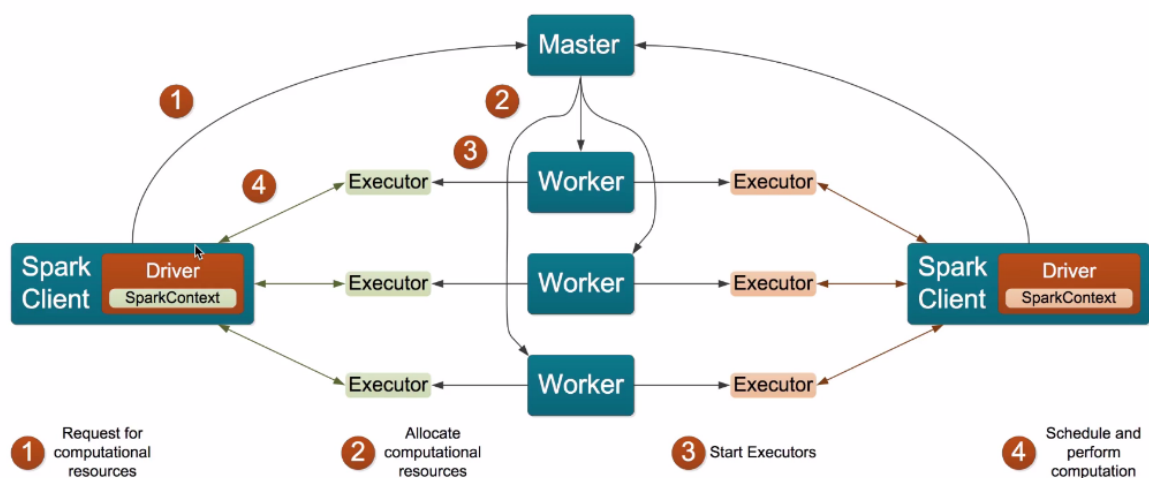


Figura 5: Arquitectura Spark

A la hora de realizar operaciones en Spark, el objeto estrella es el denominado Resilient Distributed Datasets (RDD)[11]. Se trata de una abstracción que mediante diferentes APIs disponibles para Java, Scala y Python permite manipular datos distribuidos por los diferentes nodos del clúster como si estuvieran almacenados de forma local. Este objeto es inmutable, lo cual implica que una vez creado no se le pueden añadir nuevos

elementos o eliminar los existentes, solo aplicar transformaciones y acciones sobre el.

Las operaciones que se pueden realizar sobre las RDD se agrupan, tal y como se ha adelantado antes, por transformaciones y acciones. Las primeras transforman un RDD en otro según el criterio indicado y las segundas realizan modificaciones sobre los datos almacenados en dichas RDD. Cabe destacar que las transformaciones en Spark son operaciones "lazy", lo cual implica que en realidad cada nodo memoriza la secuencia de transformaciones que ha de realizar y los procesa cuando una acción es ejecutada.

Una vez terminada la primera fase en la que los ejecutor son creados y enlazados con el driver, éste último empieza a analizar la estructura del código y genera un grafo DAG (Directed Acyclic Graph) con las operaciones que se realizan sobre la RDD. Partiendo de ese grafo genera un job por cada operación de tipo acción que encuentra y dentro de cada job separa la ejecución en diferentes stages según las dependencias que existan entre operaciones. Por último, cada stage es dividido por defecto en unidades de 64MB y a cada unidad resultante se denomina task, el cual es enviado a un ejecutor para ser procesado. El tamaño de cada task puede ser modificado programáticamente, pudiendo de esa forma manipular el número de task que un ejecutor deba ejecutar.

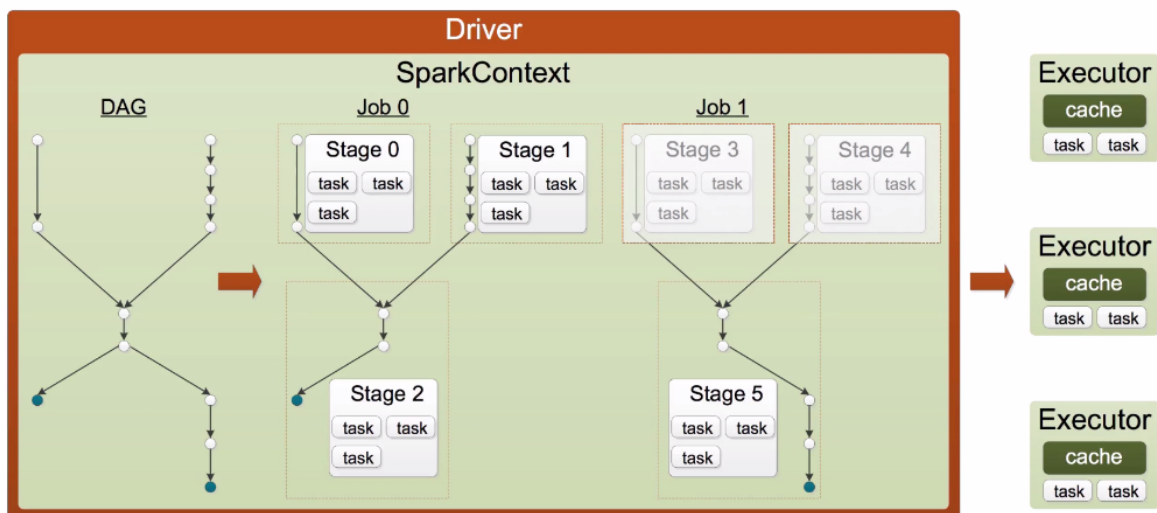


Figura 6: Arquitectura Spark

El driver, una vez habiendo recibido los resultados de todas las task que ha repartido, enviará un mensaje a los executors indicando que el procesamiento ha sido finalizado y calculará el resultado final ofreciendo la posibilidad de, por ejemplo, almacenarlo en una base de datos distribuida como Cassandra.

Bibliografía y Referencias

- [1] Avinash Lakshman y Prashant Malik. “Cassandra: a decentralized structured storage system”. En: *ACM SIGOPS Operating Systems Review* 44.2 (2010), págs. 35-40.
- [2] Raghunath Nambiar. “Towards an Industry Standard for Benchmarking Big Data Workloads”. En: ().
- [3] Sam Newman. *Building Microservices*. .ºReilly Media, Inc.", 2015.
- [4] Tilmann Rabl y col. “Solving big data challenges for enterprise application performance management”. En: *Proceedings of the VLDB Endowment* 5.12 (2012), págs. 1724-1735.
- [5] *The State of Big Data in the Large Corporate World*. Big Data Executive Survey 2013 Boston. 2013.
- [6] Matei Zaharia y col. “Spark: cluster computing with working sets.” En: *HotCloud* 10 (2010), págs. 10-10.